



Taming ROP on Sandy Bridge

Using Performance Counters to Detect
Kernel Return-Oriented-Programming

Georg Wicherski

- Senior Security Researcher at CrowdStrike, Inc.
 - x86 & ARM low-level stuff (bad at MIPS)
 - Reverse Engineering, Malware analysis
 - Exploitation and Mitigation research
- @ochsff on Twitter
- <http://blog.oxff.net/>



Introduction & Prerequisites

Return Oriented Programming

- Academic generalization / reinvention of ret2libc
 - Chain small *Gadgets* ending in ret to implement payload
 - Circumvention of NX / XN mitigations and Harvard architecture

```
pop esi  
pop edi  
ret
```



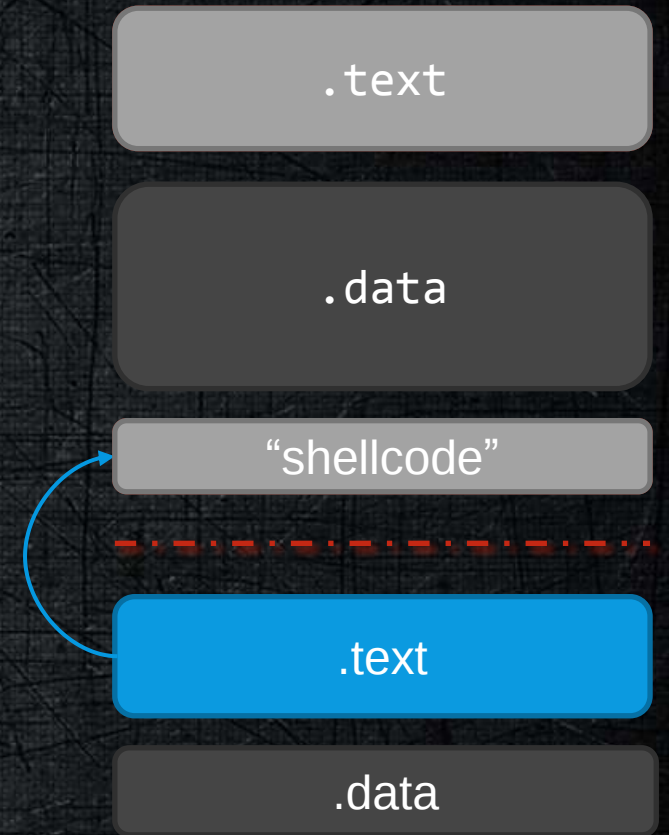
```
mov ecx, 0x100  
ret
```



```
rep movsd  
ret
```


ROP in Kernel Context

- Today: jump to executable user-space page
 - Hardware mitigation for x86 SMEP with Haswell
 - Existing software mitigation in Linux by PAX' UDEREF
- Kernel ROP already a necessity on iOS
- Expecting kernel ROP for x86 on Linux
 - SMEP patches already in mainline



Intel Performance Counters

“When the CPU utilization does not tell you the utilization of the CPU”

- Count various Performance Events in Hardware
 - Cache hits and misses
 - Special instructions executed
 - Branch prediction related events
- Present on *all* newer Intel CPUs
- Signal counter overflow with interrupt

Sandy Bridge Return Prediction

- Predicting indirect branches is important for performance
- Sandy Bridge maintains a 16 entry shadow stack of call-sites / return addresses
 - Entirely hidden from program, part of branch prediction unit
 - Only recognizes `call` / `ret` patterns (no push awareness, etc.)
- ROP naturally results in return mispredicts!

Best Friend 0x8889

- 0x89 – BR_MISP_EXEC.*: mispredicted executed branches
- 0x800 – .RETURN_NEAR: normal, near ret
- 0x8000 – .TAKEN: unconditional branch

➤ Counts mispredicted returns executed!



Related Work

Security breaches as PMU deviation:

Detecting and identifying security attacks using performance counters

- Machine-learning-like approach on Performance Counters
 - Uses a multitude of different generic counters
 - No evaluation of false positives etc. in this part
- Uses Debug-Store backed Last-Branch-Records
 - Every branch results in a multi-word memory write
- Performance evaluation only on analysis component
 - Ignores performance overhead of DS backed LBR
 - No code relased, cannot reproduce

Mitigating ROP via Last Branch Recording

- BlueHat 1st Prize, uses Last Branch Records
 - Seen in multiple places before, e.g. “Down to the Bare Metal: Using Processor Features for Binary Analysis”
- Uses MSR backed LBR storage for speed
 - Only supports storing the 8/16 last branches on the most recent dual-core CPUs
 - Checks injected into API call hooks (w/ kernel call)
- Good performance, easily circumvented

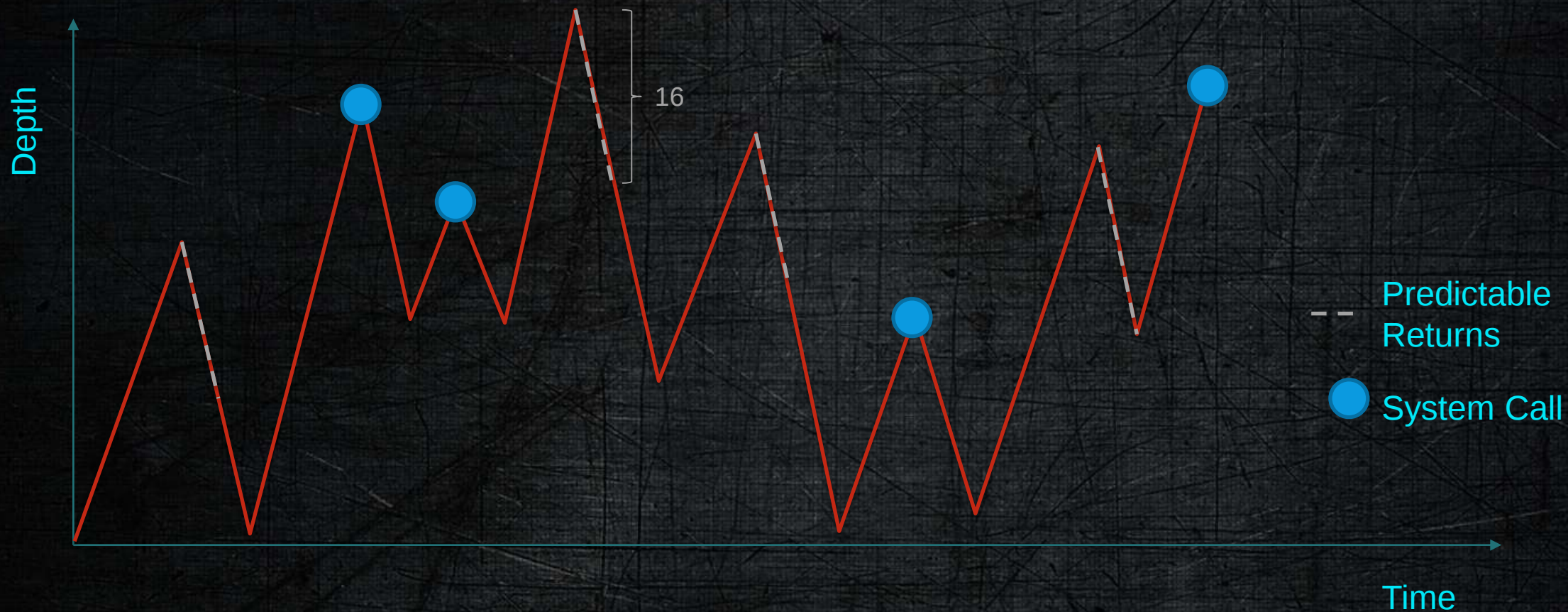
Practical Timing Side Channel Attacks Against Kernel Space ASLR

- Uses CPU cache timing attacks to break ASLR
 - Deliberately cause traps and then check performance accessing aliased cache lines
 - Suggests some solutions but some are hard to implement
- Most operating systems did not even fix all the info leaks
 - Windows and Linux make it hard *not* to obtain kernel pointers
- KASLR is broken, we need a better mitigation

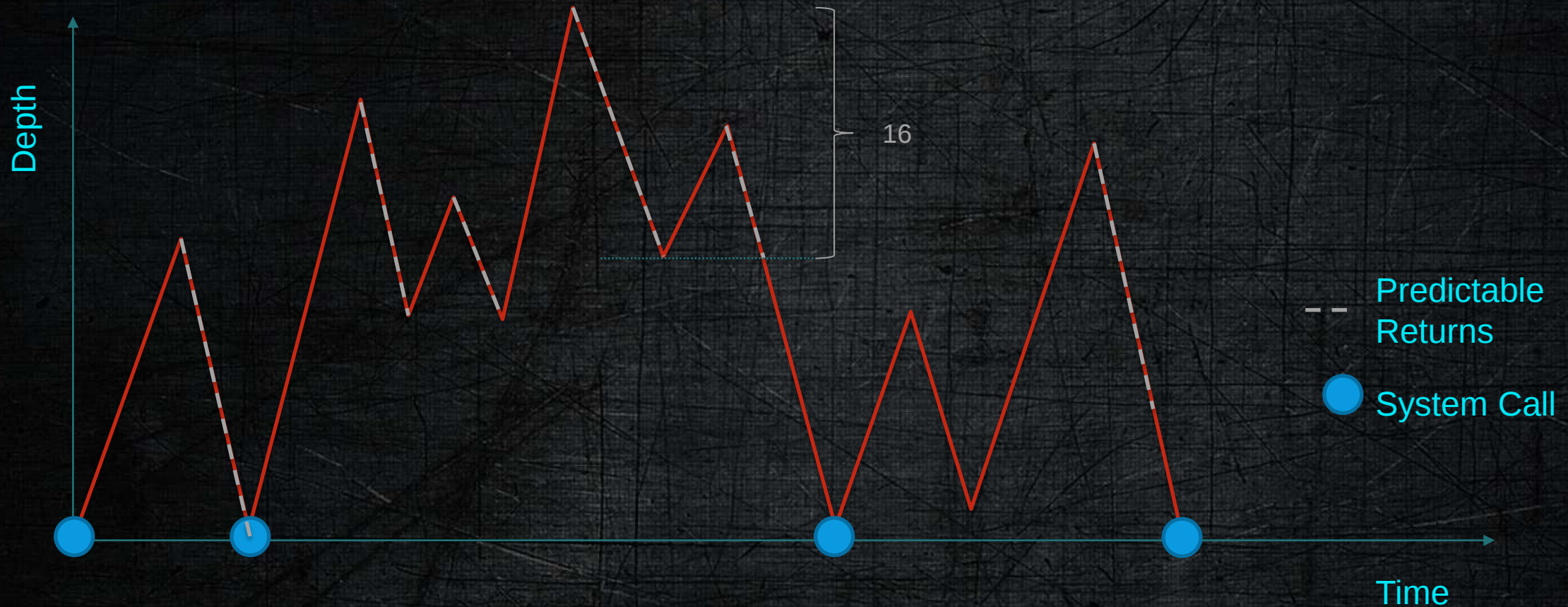


Detecting ROP with PMCs

User-Space Call Stack

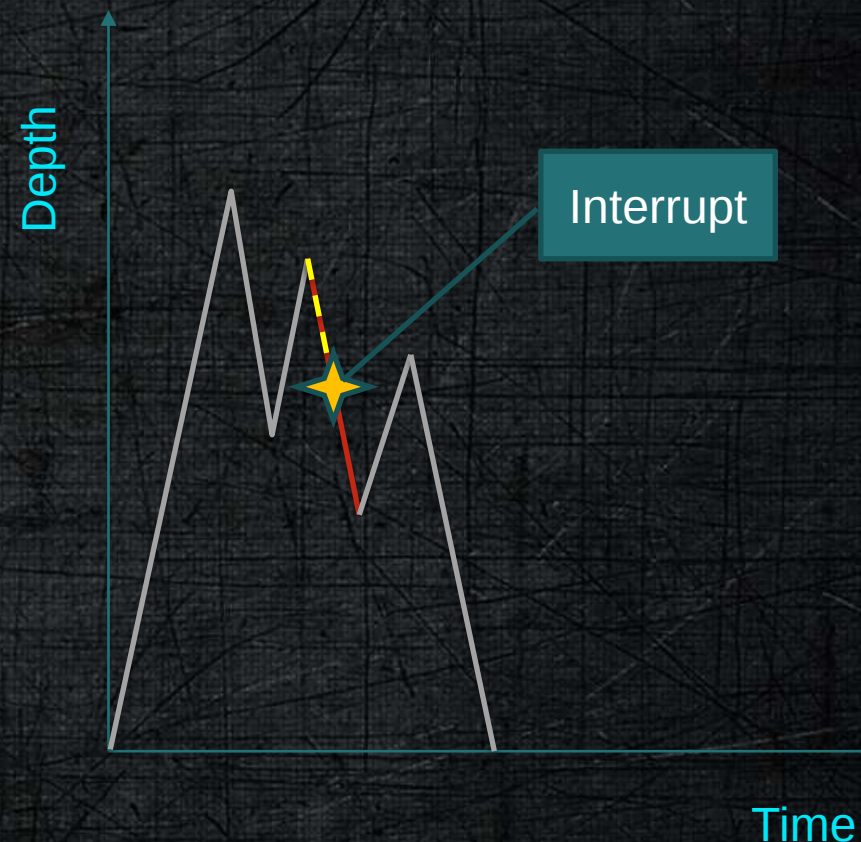


Kernel-Space Call Stack

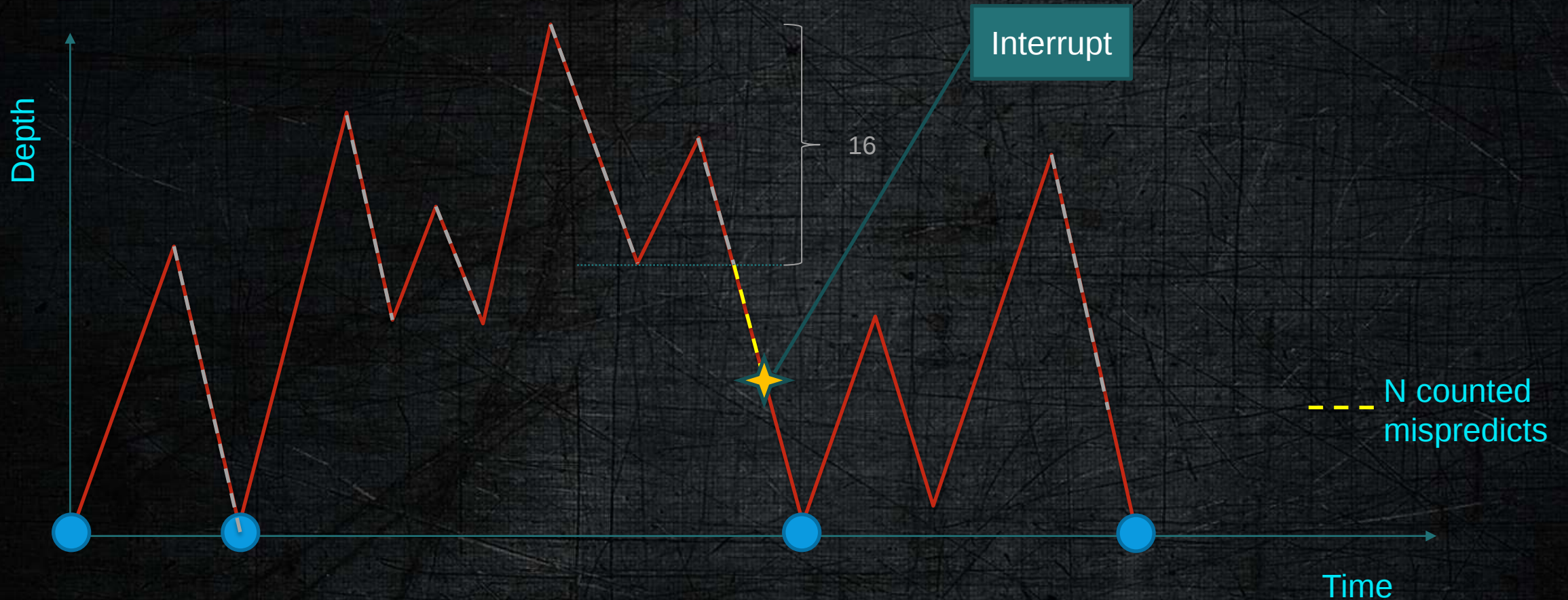


Kernel-Space ROP

- Initialize counter to value close to overflow
 - e.g. -8, because -1 interrupts on every legitimate mispredict
- “Detect!” ROP in interrupt handler if not legitimate mispredict



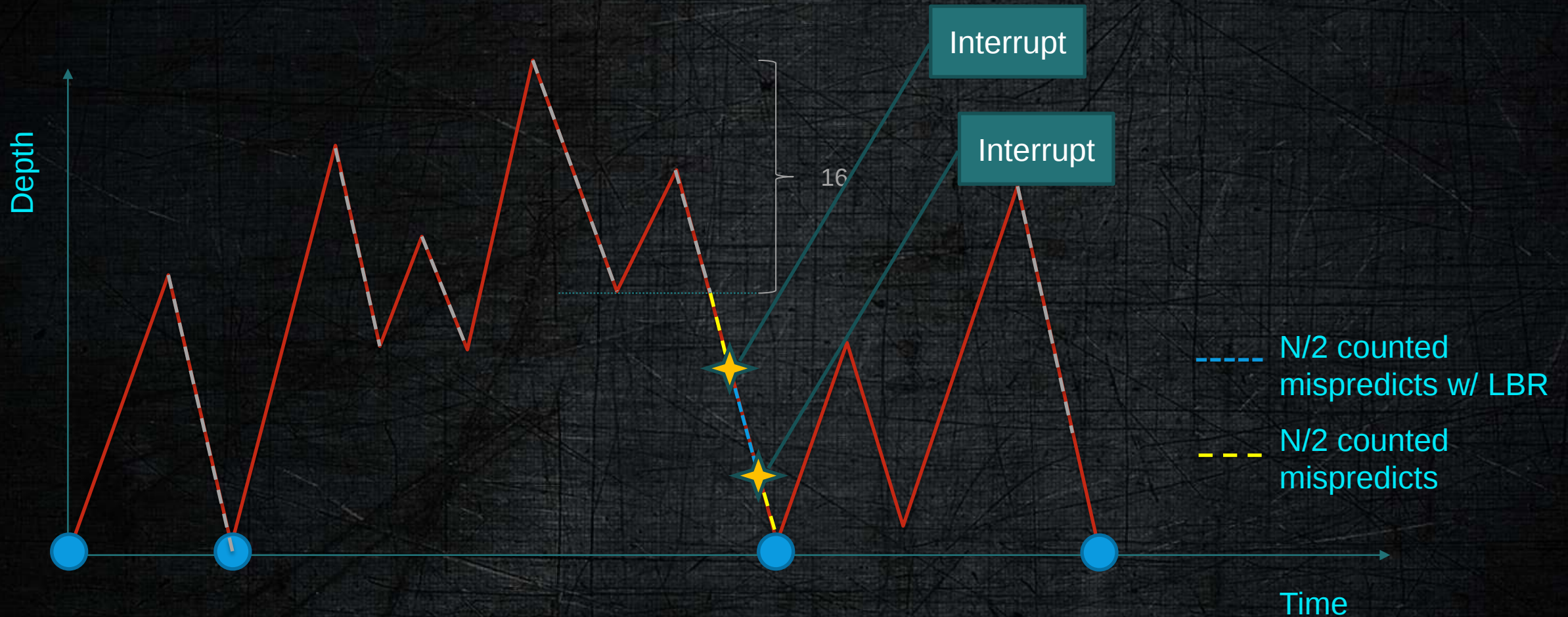
Kernel-space Call Stack w/ PMC



Differentiating Mispredicts

- Use MSR Last Branch Records to get precise but performant information about the last 16 returns
 - This includes a bit indicating mispredicts
 - Address returned from (a ret instruction)
 - Address returned to
- Check address returned to for a preceding call
 - `call rel32` exactly 5 bytes before address
 - `call r/m32` 1 to 7 bytes before
 - Verify instruction ends exactly on address returned to

Kernel-Space Call Stack w/ PMC





Implementation & Evaluation

-grsec-ropmu

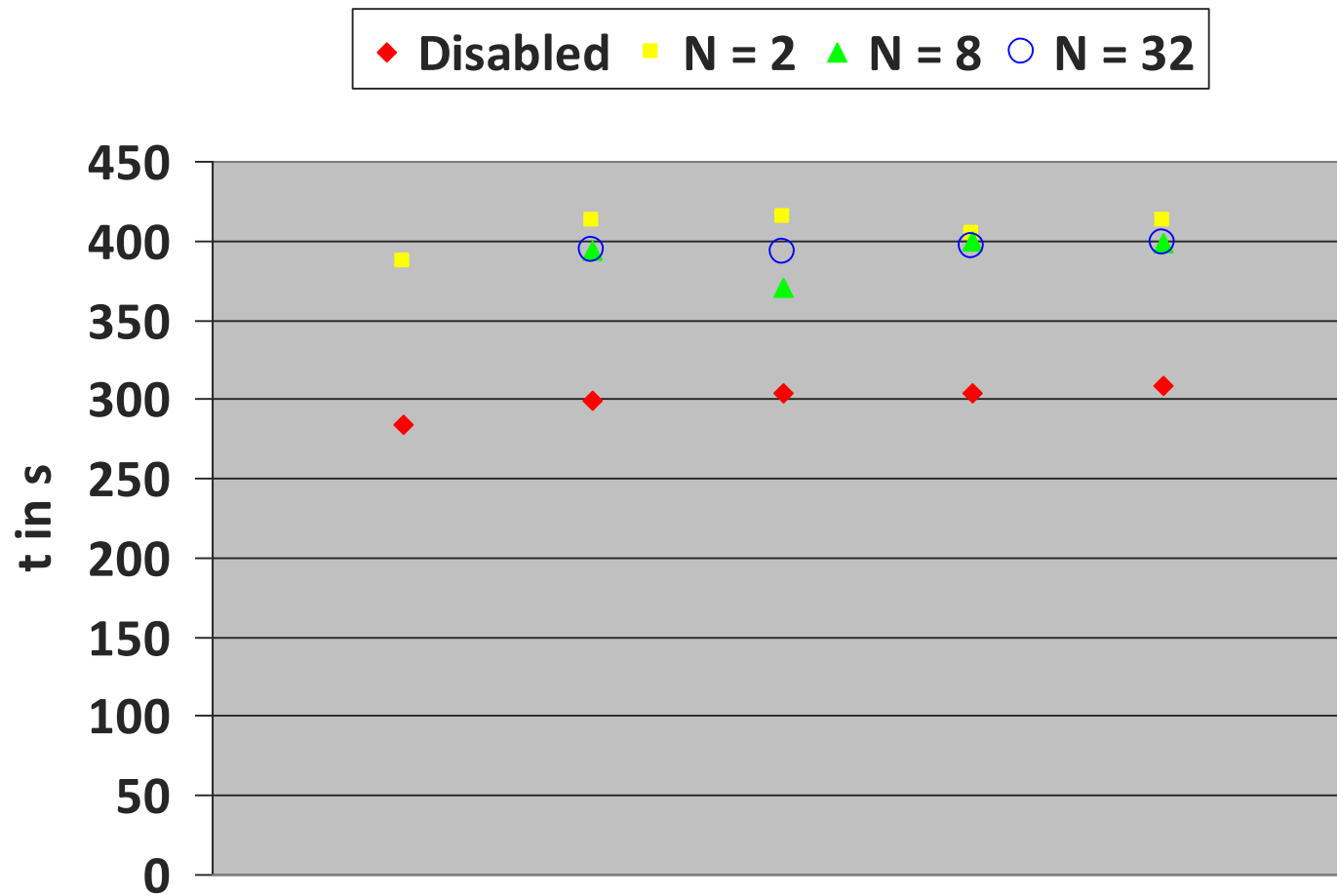
- Extension to grsecurity patch set
 - No UDEREF means no KROP required
 - Applies to 3.8.5 kernel
- Uses return mispredict counter with alternating LBR to find ROP

Performance

- Compiled ROPMU 3.8.5 kernel w/ 8 threads
 - Source on 7200 RPM volume w/ AES-NI LUKS
 - Intel(R) Core(TM) i7-2720QM CPU @ 2.20GHz

N	System	Elapsed Wall-Clock
2	135.42%	104.44%
8	128.02%	103.68%
32	130.2%	103.65%

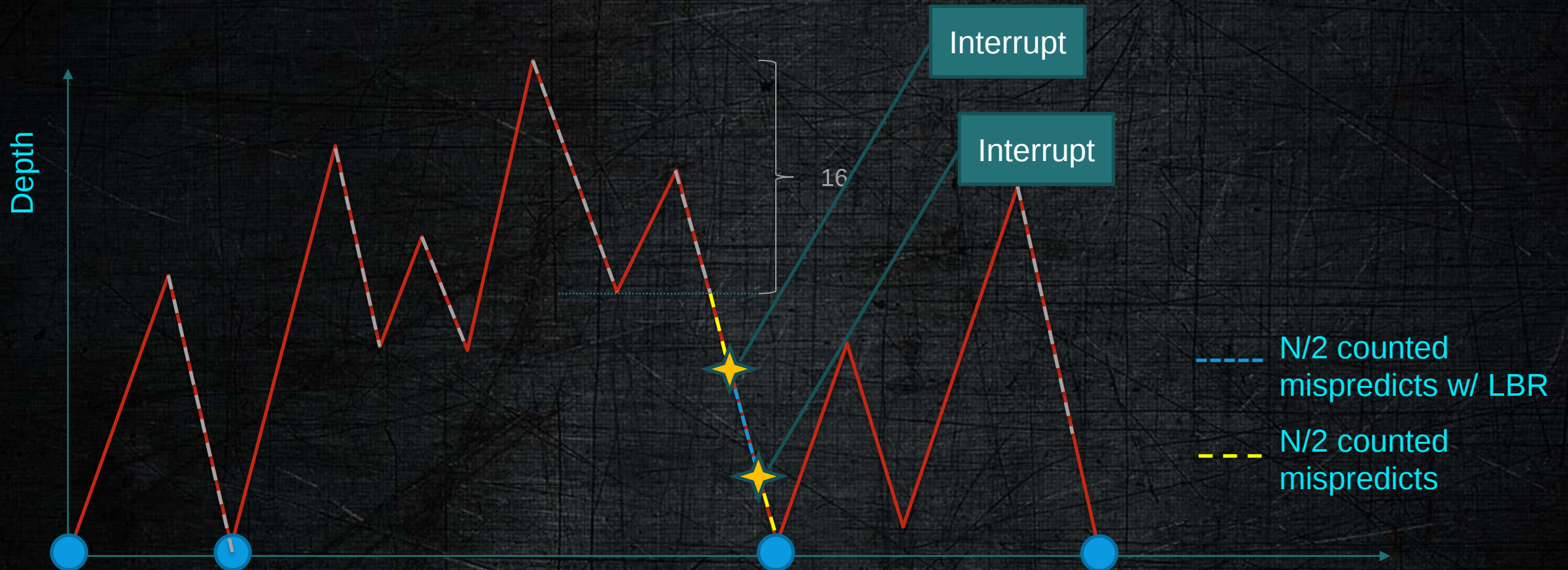
Performance – System Time





Future Work: Circumvention

c1i (ideally in Pivot)



cli Countermeasures

- Detection: Check counter value for positive signum at system call exit
 - Damage has been done already
- Use gcc plugin to instrument cli emission in kernel
 - Inadvert cli in “misaligned” instruction; just fa
 - Luckily, ff ff is invalid (ff is inc/dec Group)
 - Are there any good cli Pivots today?
- Does not work against ROPMU prototype!
 - (RO)PMU uses Non-Maskable Interrupt Vector (?)



CROWDSTRIKE